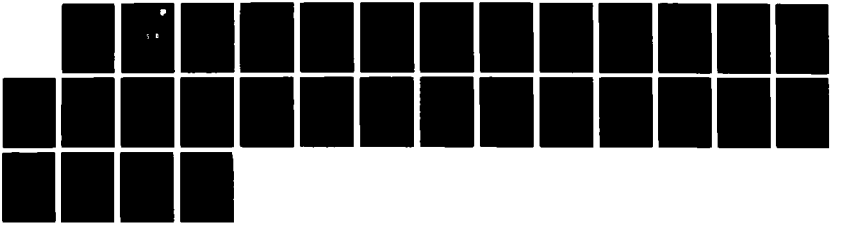
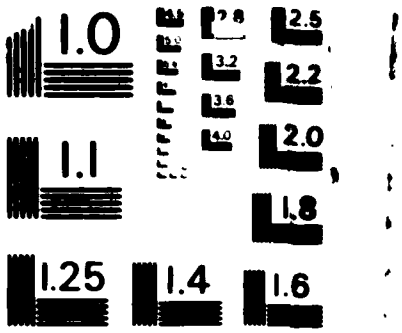
AD-A179 474 OPTIMAL PARTITIONING AND REDUNDANCY REMOVAL IN COMPUTING PARTIAL SUMS(U) STATE UNIV OF NEW YORK AT BUFFALO AMHERST A T FAM MAR 87 RADC-WP-87-5 1/1

UNCLASSIFIED F30602-81-C-0185 F/G 9/2 NL





MICROCOPY RESOLUTION TEST CHART

RADC-NP-87-5
Final Technical Report
March 1987

OPTIMAL PARTITIONING AND REDUNDANCY REMOVAL IN COMPUTING PARTIAL SUMS

University of Buffalo

Adly T. Fam

ROME AIR DEVELOPMENT CENTER
Air Force Systems Command
Griffiss Air Force Base, NY 13441-5700

Unedited version of this report, RADC-NP-87-5 dated March 1987 is being sent to the Defense Technical Information Center (DTIC) for archiving and subsequent referral through the DTIC Technical Report Data Base.

RADC-NP-87-5 has been reviewed by the RADC Public Affairs Office (PA) and is releasable to the National Technical Information Service (NTIS). At NTIS it will be releasable to the general public, including foreign nations.

APPROVED: Matthew R. Vitallo

MATTHEW R. VITALLO
Project Engineer

Accesion For	
NTIS CRA&I	☒
DTIC TAB	☐
Unannounced	☐
Justification	
By	
Distribution /	
Availability Codes	
Dist	Avail and / or Special
A-1	

APPROVED: R. L. Rhame

ROBERT L. RHAME, COLONEL, USAF
Director of Communications

FOR THE COMMANDER: Richard W. Pouliot

RICHARD W. POULIOT
Directorate of Plans & Programs

REPORT DOCUMENTATION PAGE

Form Approved OMB No. 0704-0188

1a. REPORT SECURITY CLASSIFICATION: UNCLASSIFIED	1b. RESTRICTIVE MARKINGS: N/A
2a. SECURITY CLASSIFICATION AUTHORITY: N/A	3. DISTRIBUTION/AVAILABILITY OF REPORT: Approved for public release; distribution unlimited.
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE: N/A	
4. PERFORMING ORGANIZATION REPORT NUMBER(S): N/A	5. MONITORING ORGANIZATION REPORT NUMBER(S): RADC-NP-87-5
6a. NAME OF PERFORMING ORGANIZATION: University of Buffalo; 6b. OFFICE SYMBOL (If applicable):	7a. NAME OF MONITORING ORGANIZATION: Rome Air Development Center (DCCD)
6c. ADDRESS (City, State, and ZIP Code): State University of New York, Buffalo NY 14260	7b. ADDRESS (City, State, and ZIP Code): Griffiss AFB NY 13441-5700
8a. NAME OF FUNDING/SPONSORING ORGANIZATION: Rome Air Development Center; 8b. OFFICE SYMBOL (If applicable): DCCD	9 PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER: F30602-81-C-0185
8c. ADDRESS (City, State, and ZIP Code): Griffiss AFB NY 13441-5700	10 SOURCE OF FUNDING NUMBERS: PROGRAM ELEMENT NO. 61102F; PROJECT NO. 2305; TASK NO J8; WORK UNIT ACCESSION NO. PF

11. TITLE (Include Security Classification)

OPTIMAL PARTITIONING AND REDUNDANCY REMOVAL IN COMPUTING PARTIAL SUMS

12. PERSONAL AUTHOR(S)
Adly T. Fam

13a. TYPE OF REPORT	13b. TIME COVERED	14. DATE OF REPORT (Year, Month, Day)	15. PAGE COUNT
Final	FROM Dec 85 TO Jan 87	March 1987	31

16. SUPPLEMENTARY NOTATION
N/A

17. COSATI CODES		
FIELD	GROUP	SUB-GROUP
12	07	
09	01	

18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
Partial Sums, Redundancy Removal, Optimal Partitioning, Concurrent Computing

19. ABSTRACT (Continue on reverse if necessary and identify by block number)

Two novel algorithms for simultaneous computation of a large number of partial sums are introduced, their performance assessed, and architectures for their implementation suggested. The direct computation of D operations are replaced by O(D/ log D). The new approach is based on a new concept of optimal partitioning and redundancy removal in arithmetic intensive, high throughput computing that is expected to be the basis of a new class of algorithms which represent a departure from brute force parallel computation where inherent redundancy is not detected or removed.

20 DISTRIBUTION/AVAILABILITY OF ABSTRACT: ☐ UNCLASSIFIED/UNLIMITED ☒ SAME AS RPT ☐ DTIC USERS	21 ABSTRACT SECURITY CLASSIFICATION: UNCLASSIFIED
22a NAME OF RESPONSIBLE INDIVIDUAL: Matthew R. Vitallo	22b TELEPHONE (Include Area Code): (315) 330-3226; 22c OFFICE SYMBOL: RADC (DCCD)

DD Form 1473, JUN 86 — Previous editions are obsolete

I. INTRODUCTION

In this paper two dual algorithms for the efficient concurrent computation of partial sums are introduced and their performance assessed. These algorithms are members of a novel class of algorithms and architectures that is particularly suited for arithmetic intensive, high throughput computing. This class is based on partitioning the desired computations into parts that can assume a relatively small number of distinct forms. The redundancy resulting from the appearance of a given form more than once is removed by computing each form only once. The computation of all the distinct forms is performed first, and then combined appropriately to obtain the desired results. The partitioning size is parameterized by a partition parameter, or parameters. A cost function is defined to take into account all the relevant factors such as the number of operations, chip area, and computation time. This cost function is, after partitioning, a function of the partition parameter(s) with respect to which it could be minimized. The minimum cost is attained at certain optimal partition sizes, that should be used in the implementation.

The partial sums computation of interest to us in this paper are expressed in the form

$$Y = BX + U \qquad (1.1)$$

where the U, X, B, and Y are of dimensions Mx1, Nx1, MxN, and Mx1, respectively. U and X are data vectors, and B is a binary matrix of zeros and ones that define the desired partial sums Y. The vector X contains all the data from which subsets for the

various partial sums are selected. The vector U is added only to complete the duality of the two algorithms, and has an insignificant effect on their behavior.

The first algorithm to compute these partial sums is based on partitioning the output, Y. In section II the preliminary version of this algorithm presented in [1, 2] is completed. A dual algorithm based on partitioning X, the input, is presented in section III where the complete duality of the two algorithms is established. In assessing the performance of the algorithms, some parameters are treated initially as continuous, while only integer values could be used in the implementation. This is shown in section IV to have an insignificant effect on the performance of the algorithms. In section V various aspects of implementing the algorithms are considered, with particular emphasize on parallel and pipelined architectures for high throughput applications.

Let the number of operations in direct computation of Y be denoted by D. The proposed algorithms are shown to result in replacing D by $O(D/\log D)$. The proportionality factor associated with this "order of" estimate is in the range 1 to 4 for various instances of the algorithms, as well as for their combination as shown in section VI. Generalizations of this work are suggested also in section VI, while suggestions for applications and topics for further research conclude the paper in section VII.

II. ALGORITHM BASED ON OUTPUT PARTITIONING

In this section we discuss an algorithm that computes a set of concurrent partial sums by partitioning them optimally into a number of subsets. Each subset is computed independently, applying the concept of redundancy removal which is best introduced first through the following characteristic case, essential to the algorithm.

A. First Characteristic Case

Consider the computation of the partial sums of (1.1)

$$Y = BX + U \tag{2.1}$$

for B with dimensions r x n, where

$$n = 2^r - 1 \tag{2.2}$$

In addition, all the columns of B are distinct and none of all zero entries. This implies that the entries of each columns of B corresponds to the binary representation of one of the integers $\{1, 2, \ldots, 2^r - 1\}$. Also, each row of B contains exactly 2^{r-1} ones and $2^{r-1} - 1$ zeros.

The algorithm is comprised of two steps that are applied alternately until all partial sums are computed:

<u>Step 1</u>

Compute one of the partial sums. This requires $A_{11} = 2^{r-1}$ additions and eliminates one row from B, and the corresponding entry of U. There are now two identical columns of B corresponding to each of the binary representations of the numbers $\{1, 2, \ldots, 2^{r-1} - 1\}$ and one column with all zero entries.

Step 2

Remove the zero column from B and the corresponding entry from X. Remove one of each two identical columns of B after adding the corresponding entries of X. This requires $A_{12} = 2^{r-1} - 1$ additions.

The two steps comprise one iteration of the algorithm. The first execution of the algorithm requires $A_1 = A_{11} + A_{12} = 2^r - 1$ and replaces r by r - 1. The ith execution requires $A_{i1} = 2^{r-i}$, $A_{i2} = 2^{r-i} - 1$ additions for steps 1 and 2 respectively, resulting in

$$A_i = A_{i1} + A_{i2}$$
$$= 2^{r-i+1} - 1, \quad i = 1, 2, \ldots, r. \tag{2.3}$$

From eq. (2.2, 2.3) the total number of additions to compute all partial sums is found to be

$$A(n) = \sum_{i=1}^{r} A_i$$
$$= 2(2^r - 1) - r = 2n - r$$
$$= 2n - \log(n+1) = O(n) \simeq 2n \tag{2.4}$$

where log is to the base 2 throughout this paper. The number of additions per output is then

$$C(n) = (2(2^r - 1) - r)/r = 2(n/r) - 1$$
$$= 2(n/\log(n+1)) - 1 = O(n/\log n) \simeq 2n/\log n \tag{2.5}$$

Since each row of B contains 2^{r-1} ones in this characteristic case, a direct computation of each partial sum independently results in the following number of additions per output

$$D(n) = 2^{r-1} = (n + 1)/2 = O(n) \simeq n/2 \tag{2.6}$$

The efficiency of our approach in comparison to direct

computation could be expressed by the ratio

$$\eta(n) = C(n)/D(n) = 4(n - (\log(n+1))/2)/(n+1)\log(n+1)$$
$$= O(1/\log n) \simeq 4/\log n \qquad (2.7)$$

or equivalently by the expressions

$$\eta(n) = O(1/\log D(n)),$$
$$C(n) = O(D(n)/\log D(n)) \qquad (2.8)$$

which indicate the type of computational savings achieved by the proposed algorithm.

An important aspect of this algorithm is its invariance with respect to the partitioning within the r outputs in steps 1 and 2 above. So, instead of computing one output at a time, followed by merging inputs corresponding to redundant computation for the remaining outputs, we could partition the r outputs into two sets of r_1 and r_2 outputs each. Due to the assumption of distinct columns of B, and the carefully chosen n according to (2.2), the computations involved in the r_1 and r_2 outputs are mutually exclusive and could be computed independently. The r_1 outputs require $2^r - 1 - (2^{r_1} - 1)$ additions to add inputs corresponding to identical columns of B_1, followed by $A(n_1)$ additions as determined by the above characteristic case, but with dimensions n_1 and r_1. Similar argument for the remaining r_2 outputs lead to

$$A(n) = A(n_1) + 2^r - 1 - (2^{r_1} - 1) +$$
$$A(n_2) + 2^r - 1 - (2^{r_2} - 1) +$$

which, with $r = r_1 + r_2$ lead to

$$A(n) = 2n - r$$

identically to (2.4). For n a power of 2, a scheme of progressively finer partitioning, and combining data with

identical roles in the required computation is possible here. At every step, the number of parallel computations doubles until all outputs appear simultaneously at the last step. In the fastest implementation of an adder tree to be discussed in the sequel, and regardless of the partitioning method, and if all data additions are performed by adder trees, then all the outputs will be computed after a time period corresponding to r - 1 additions.

The dimensions of this characteristic case were carefully chosen. Therefore, only the redundancy removal aspect of the algorithm was applied. In the sequel we consider problems with arbitrary dimensions and show how redundancy removal is combined with optimal partitioning to result in a complete application of the algorithm.

B. The General Case

To apply the above approach to the general case with arbitrary dimensions (1.1), we partition the M partial sums Y, and similarly U, into s sets Y_i, i = 1, 2, ..., s, of r partial sums each. The parameters r and s are related by

$$M = rs \tag{2.9}$$

Furthermore, we assume that r satisfies

$$N \geq 2^r - 1 \tag{2.10}$$

For a worst case analysis, we assume that all the distinct $2^r - 1$ nonzero binary vectors are present in each B_i. For each group of r partial sums we begin by executing step 2 as introduced above. This requires $N - (2^r - 1)$ additions per B_i. The problem is now identical to the characteristic case discussed above, and therefore $2(2^r - 1) - r$ additions per r partial sums are needed

to complete the computation. So the number of additions per r partial sums is, in this worst case analysis

$$A(N,r) = (N + 2^r - r - 1) \qquad (2.11)$$

Equivalently, the number of additions per output is

$$C(N,r) = (N + 2^r - r - 1)/r$$

$$= (N + 2^r - 1)/r - 1 \qquad (2.12)$$

There are at least two approaches to investigate the optimal values of r at which C(N,r) attains its minimum. In the first approach the minimum is found by treating r as a continuous variable. The derivative of C with respect to r vanishes at

$$N = 2^r(r \ln 2 - 1) + 1 \qquad (2.13)$$

at which C attains its minimum value. There is no explicit closed form expression of r as a function of N that could be obtained from eq. (2.13). However, such an expression is not essential in applying the algorithm where only integer values of r are of interest. The first integer value of r for which (2.10, 2.13) are simultaneously valid is 3. A table of the values of N corresponding to r = 3, 4, ... could be formed to cover the range of values of N of interest. The initial estimate

$$r_1 = \log(N-1) \qquad (2.14)$$

could be used either to identify the range of values of r that are then used to generate a small table via (2.13), or as the initial term in the iteration

$$r_{i+1} = \log(N-1) - \log(r_i \ln 2 - 1), \quad i = 1, 2, \ldots \qquad (2.15)$$

which converges rapidly to the solution of (2.13). With r_1 as defined in (2.14), the next two iterations result in

$$r_2 = \log(N-1) - \log(\ln(N-1) - 1),$$

$$r_3 = \log(N-1) - \log(\ln(N-1) - \ln(\ln(N-1) - 1)) \qquad (2.16)$$

From (2.12, 2.13) we obtain the minimum number of additions per output corresponding to the optimal value of r

$$C = ((N-1)\ln 2 / (r \ln 2 - 1)) - 1 \qquad (2.17)$$

which with (2.14, 2.15) result in

$$C \simeq ((N - 1) / (\log(N-1) - 1)) - 1 \simeq N / \log N \qquad (2.18)$$

Direct computation requires an average of

$$D = N/2 \qquad (2.19)$$

additions per partial sum. The efficiency of the proposed algorithm is characterized by

$$\eta = C/D \simeq 2/\log N \qquad (2.20)$$

which is a conservative estimate of its performance, since we are comparing our worst case to the average direct computation. In this algorithm, all the blocks of r partial sums are computed independently. Redundancy is removed only within each block. Further redundancy could be removed based on computation shared between blocks. This additional redundancy is insignificant and its removal would require complicated communication schemes. A second approach in investigating the optimal values of r is presented in section IV, where r and N are assumed to be integers.

III. A DUAL ALGORITHM

In this section we introduce an algorithm based on input partitioning. This algorithm is a dual to the one discussed above with the input and the output roles exchanged in all significant expressions and statements. There are, however, differences in implementing the two algorithms. There is also a subtle difference in generalizing the two algorithms to operations other than addition. These differences will be addressed in the following sections.

The following characteristic case is of essence since it illustrates the redundancy removal aspect of this dual algorithm.

A. Second Characteristic Case

Consider the computation of the partial sums of (1.1)

$$Y = BX + U \tag{3.1}$$

but for B with dimensions n x r, where

$$n = 2^r - 1 \tag{3.2}$$

In addition, all the rows of B are distinct and none of all zero entries. This implies that the entries of each row of B correspond to the binary representation of one of the integers $\{1, 2, \ldots, 2^r - 1\}$. Also, each column of B contains exactly 2^{r-1} ones and $2^{r-1} - 1$ zeros. B is, in this case, the transpose of that in the first characteristic case. Let us first ignore U. Computing Y in this case amounts to computing all the partial sums of the entries of X.

Let P(r) be the number of additions required to compute all the $2^r - 1$ nonzero partial sums of r elements of a set. If one

more element is added to the set, then the best that could be done is to add the new element to each of the existing partial sums. This requires $2^r - 1$ additions. This is the smallest number of additions needed to generate all the additional partial sums that include the new element. The result is the following recursion

$$P(r+1) = P(r) + (2^r - 1) \tag{3.3}$$

which results in

$$P(r) = 2^r - r - 1 \tag{3.4}$$

From eq. (3.2, 3.4) the total number of additions to compute all partial sums is found to be, after including n additions for U

$$A(n) = 2(2^r - 1) - r = 2n - r$$

$$= 2n - \log(n+1) = O(n) \simeq 2n \tag{3.5}$$

which is identical to (2.4), but with r and n indicating the number of columns and rows of B respectively in this dual case. The above result could also be obtained if the r inputs are partitioned into two subsets of r_1 and r_2 inputs each, respectively. All the partial sums of each subset are computed, and the results combined to obtain Y. Regardless of the method of partitioning, the fastest implementation would result in all the outputs after a time corresponding to r - 1 additions, with some components of Y computed even sooner. The number of additions per input is then

$$C(n) = (2(2^r - 1) - r)/r = 2(n/r) - 1$$

$$= 2(n/\log(n+1)) - 1 \simeq 2n/\log n \tag{3.6}$$

Direct computation of the partial sums requires a number of additions that is equal to the number of ones in B. This is easy to see, since each row of B corresponds to a number of additions

of entries of X that is one less than the number of ones in that row. An extra addition is required to include the corresponding element of U, thus resulting in one element of Y. Since B in this characteristic case is the transpose of that of the first characteristic case, it has the same total number of ones of $r2^{r-1}$. The number of additions per input is, therefore

$$D(n) = 2^{r-1} = (n + 1)/2 \simeq n/2 \qquad (3.7)$$

which is identical to (2.6), but with the roles of the rows and columns interchanged.

B. The General Case

To apply the above approach to the general case with arbitrary dimensions (1.1), we need to combine the redundancy removal aspect of the algorithm as introduced above with optimal partitioning. Let the N inputs of X be partitioned into s sets X_i, $i = 1, 2, \ldots, s$, of r elements each. The parameters r and s are related by

$$N = rs \qquad (3.8)$$

Furthermore we assume that r satisfies

$$M \geq 2^r - 1 \qquad (3.9)$$

To compute all the partial sums of each of the s sets of entries of the X_i's, a total of $(2^r - r - 1)N/r$ additions are needed. This follows from (3.4) and (3.8). Each partial sum, corresponding to one entry of Y, could then be obtained at the cost of N/r extra additions. This includes the proper entry of U. The total of the extra additions to obtain Y is NM/r. The total additions to compute all partial sums is then

$$A(M,N,r) = (2^r - r - 1)N/r + MN/r \qquad (3.10)$$

and the number of additions per r inputs is

$$A(M,r) = (M + 2^r - r - 1) \qquad (3.11)$$

which is the dual of (2.11) with M replacing N. Equivalently, the number of additions per input is

$$C(M,r) = (M + 2^r - r - 1)/r$$

$$= (M + 2^r - 1)/r - 1 \qquad (3.12)$$

which is also the dual of (2.12). The above establishes the complete duality of the two algorithms. The remainder of our analysis is identical, via duality and proper exchange of variables, to that of section II. Comments on redundancy between blocks are similar to those made at the end of section II, but with the roles of N and M interchanged.

IV. EFFECT OF INTEGER PARAMETERS

There are several effects of restricting the parameters N, M, and r to integer values. We will be concerned mainly with the first algorithm of output partitioning, since duality extends the results immediately to the second algorithm. Let us ivestigate first the effect of restricting r and N to be integers. Equation (2.12) could be rewritten in the form

$$C(N,r) = N/r + (2^r - r - 1)/r \qquad (4.1)$$

This could be viewed as a straight line function of N with a slope of 1/r and a displacement of $(2^r - r - 1)/r$, both of which are parameterized by r. Let us generate these straight lines for r = 1, 2, ... The lowest upper bound of this collection of straight lines is a piecewise linear curve, each segment of which is a part of one of the above straight lines that corresponds to

a particular integer value of r as depicted in Fig. 1. This piecewise linear curve represents the least possible number of additions per output C as a function of just N, since r was used as a parameter in generating this lowest upper bound. The vertices of the resulting piecewise linear curve are the points at which $C(N,r) = C(N,r+1)$, and occurs at N_r where

$$N_r = 2^r(r-1) + 1 \tag{4.2}$$

The range of values of N for which a given value of r should be used is $N \in [N_{r-1}, N_r]$. This range, followed next by the related difference and ratio

$$2^{r-1}(r-2) + 1 \leq N \leq 2^r(r-1) + 1,$$

$$N_r - N_{r-1} = r\,2^{r-1},$$

$$N_r/N_{r-1} = (2^r(r-1) + 1)/(2^{r-1}(r-2) + 1) \simeq 2 \tag{4.3}$$

should be used in assessing the value, or range of values of r that correspond to the range of values of N in a given application or problem.

One way of relating the optimization of r as a continuous variable to that of integer r is as follows. Every segment of the piecewise linear curve defined above is a tangent to the curve of minimal C as a function of N that results from treating r as a continuous variable. The two curves are farthest apart at N_r, r = 1, 2, ... At these points we get from (4.1, 4.2)

$$C = 2^r - 1 = (N_r - 1)/(r-1) - 1$$

$$= O(N/\log N) \simeq N/\log N \tag{4.4}$$

which exhibits the same asymptotic behavior as for continuous r. This indicates the insensitivity of the optimal performance to small changes in r.

The effect of restricting N to be integer values is simply incorporated by considering only those points on the above piecewise linear curve that correspond to integer N.

The last issue of importance here concerns M and s. It is obvious that the optimization makes sense only if $M \geq r$ in the first algorithm, and $N \geq r$ for the dual one. Other than that, if s is not an integer, then all the partitioned parts of the problem will not be of the same size. In this case we simply use one of the nearest integer values to s, and only minor deviation from the optimal behavior should result. We can limit the difference to only one of the partitioned systems, or try to make them all as close as possible with a difference of one row (column in the dual case) at the most between any pair.

V. IMPLEMENTATION CONSIDERATIONS

In this section architectures that implement the above algorithms are introduced. Only the basic concept of each implementation are considered, since details are better left for individual applications.

A. The Output Partitioning Algorithm

Since every group of r partial sums is evaluated independently, it suffices to consider the implementation of one such group. A parallel architecture is then obtained by replicating this implementation s times. Next, two types of implementation are discussed.

If the data is obtained sequentially, then a pipelined architecture is particularly suitable. We examine here an

architecture that implements the algorithm for only one group of r partial sums, which is then replicated in parallel or used sequentially for the complete implementation. This architecture is based on a RAM and an adder. Each data item is associated with the corresponding column of B_i which is used as an address tag. The contents of the memory at this address are read, added to the data and restored in the same location. It is clear that this simple architecture adds data corresponding to identical columns of B_i. After all the data is obtained, we begin computing the partial sums by reading all the data from locations with 1 as the last bit of their address and accumulating it, using the adder. The data is then read, the last bit of its address tag removed, and then applied to a similar memory-adder architecture but with half the previous memory size. clearly, the above implements the two steps of the algorithm. The above approach could be modified in a number of ways to adapt it to a particular environment, such as a microprocessor implementation or a particular bus architecture.

Our purpose here is to present only the basic approaches, but since high throughput applications are of particular interest, the following pipelined architecture is of particular importance.

The data is passed only once in the above memory-adder. The data read from the memory adder is obtained in the desired order of an increasing address tag. Since the data is now well ordered, and with distinct address tags, the partial sums could be computed in an adder tree that is structured to implement the

steps of the algorithm as depicted in Fig. 2. This adder tree can serve several memory-adders interfaced to it via serial in / parallel out shift registers.

B. The Input Partitioning Algorithm

This algorithm is naturally suited for parallel implementation. All the partial sums of each group of r inputs are computed independently, and each output is obtained by collecting one of the partial sums from each input group and one of the entries of U. We will consider the computation of all the partial sums of only one of the groups of r inputs, since this could be used as the building block of various implementations. An elegant structured adder tree architecture that computes all the partial sums of r numbers is based on the recursion of eq. (3.3). As illustrated in Fig. 3, the tree architecture implements the recursion directly. The three partial sums of the first two inputs are added each to the third input, to result in the desired seven partial sums. This "nesting" could be repeated as many times as needed. A parallel architecture based on copies of such a tree is obvious. A pipelined architecture is also possible, where the sets of r input are applied sequentially to one tree. Of course all the r inputs of each set are applied in parallel to the tree.

VI. COMPARISONS AND GENERALIZATIONS

In this section, we consider the performance of the above two algorithms when they are both available simultaneously. Also, generalization of the algorithms and their cost functions are discussed.

A. Performance of The Combined Algorithms

The total number of additions to compute Y of (1.1) via the output partitioning algorithm could be readily shown from the analysis in section II to satisfy

$$A_t = O(MN/\log N) \simeq MN/\log N \qquad (6.1)$$

while for input partitioning we obtain

$$A_t = O(MN/\log M) \simeq MN/\log M \qquad (6.2)$$

The performance of the two algorithms, combined, is determined by

$$A_t \simeq MN/\max\{\log N, \log M\} \qquad (6.3)$$

which results in

$$A_t \simeq 2MN/\log NM \qquad (6.4)$$

as a consequence of the obvious inequality

$$MN/\max\{\log N, \log M\} < 2MN/(\log N + \log M) = 2MN/\log NM$$

The total number of additions required for direct computation is, in the average

$$D_t = MN/2 \qquad (6.5)$$

The efficiency of the combined algorithms is then determined by

$$A_t / D_t \simeq 4/\log NM \qquad (6.6)$$

which is achieved by using the output partitioning algorithm if $N \geq M$ and its dual, the input partitioning one if $M > N$. The two algorithms are combined here only in the sense that they are both

available. Only one of them is used, however, in any given instance as determined by the values, or range of values of M and N. Of course, for the case of M = N the two algorithms are equally effective, and only other implementation considerations could favor either.

B. Generalizations

The output partitioning algorithm is applicable to concurrent partial computations under any operation that is both associative and commutative. The addition is only one such operation. Commutativity is required since the algorithm might perform computations on the input data in other than the order in which it was given. The dual algorithm however could be applied without requiring commutativity and is therefore applicable in computing expressions in any associative operation. For example it could be used in computing partial products of a set of matrices that are not necessarily commutative. It is also possible to apply these algorithms to expressions in finite fields and rings as well as Boolean algebra ones.

The cost function used need not be restricted to the number of operations. For example in VLSI applications, a properly defined cost function might include terms to reflect area and time delay. Also several partition parameters might be present in the cost function.

VII. CONCLUSION

Two novel algorithms for simultaneous computation of a large number of partial sums are introduced and their performance assessed. The direct computation of D operations are replaced by $O(D/\log D)$. The new approach is based on a new concept of optimal partitioning and redundancy removal in arithmetic intensive, high throughput computing that is expected to be the basis of a new class of algorithms. This factor of $O(1/\log D)$ reduction in the required computation appears to be generic to the new class of algorithms and is expected to appear in their application to other problems such as vector matrix multiplication and other linear algebra operations.

The new algorithms represent a departure from brute force parallel computation where inherent redundancy is not detected or removed. The resulting architectures admit of systolic implementation in part, but also require some form of broadcasting certain computations . The arguments for systolic processing [3] are valid, but for sufficiently large problems, the removal of redundancy via the new algorithms could result in fundamental improvement. For example, a preliminary assessment of an algorithm of the new class for a parallel vector multiplier accumulator chip indicates that with VHSIC II implementation, at least three times more multipliers could be accommodated in comparison to direct implementation. The new approach is flexible and could be optimized under various cost functions, such as chip area, number of operations, time, time area product ...etc. It also could be applied on various levels; system, board ,or chip.

Since broadcasting is particularly simple in optical computing, the new algorithms could be of particular value. This could have an impact on significant applications as in [4].

Application to FIR filters, utilizing their full structure, should unify and improve the result in [5,6]. Optimization of various algorithms for large chips, combining the new algorithms with techniques of the type in [7] is also one of the objectives of this research. Since the approach is directly applicable to finite fields and rings, it could also be considered for optimal coding / decoding architectures and for computations with a variety of arithmetic systems. The applicability to Boolean expressions mentioned in section VI could result in a new approach to PLA chip design.

Finally, we propose the following topics for further research :

1- Developing algorithms and architectures of the proposed class for vector-matrix multiplication, other linear algebra operations, and for key signal processing algorithms and filters.

2- Optimizing the algorithms for software and hardware implementation, including advanced technology chip sets of VHSIC II and GaAs types as well as those utilizing optical components.

3- Assessing the performance of the new algorithms and architectures in their various implementation modes for certain supercomputer systems and for key fast signal processing applications, such as digital beam forming, target identification and other radar and communication applications.

ACKNOWLEDGMENTS

This work was supported by Rome Air Development Center contract F30602-81-C-0193 through the Southeastern Center for Electrical Engineering Education under subcontract no. SCEEE-PDP/85-0042, and contract F30602-81-C-0185, Task #C-6-2442 through Georgia Institute of Technology. The author is grateful for the continuing support and involvement of John Patti, Richard Smith, and Matthew Vitallo of RADC, Rome, New York.

REFERENCES

[1] A. Fam, "Concurrent Computing of Partial Sums via a Parallel Memory-Accumulator Architecture," Seventeenth Annual Asilomar Conference on Circuits, Systems and Computers, Pacific Grove, California, pp. 84-87, Oct. 31 - Nov. 2, 1983.

[2] A. Fam, "An Algorithm for Parallel Computation of Partial Sums, 1984 ACM Twelfth Annual Computer Science Conference, Philadelphia, Pennsylvania, pp. 131-133, February 14-16, 1984.

[3] H. Kung, "Why Systolic Architectures?", computer, Vol. 15, No. 1, pp. 37-46, Jan. 1982

[4] H. Caufield, J. Neff, and W Rhodes, "Optical Computing: The Coming Revolution in Optical Signal Processing," Laser Focus/Electro-Optics, Nov. 1983.

[5] A. Fam, "A Multi-Signal bus Architecture for FIR Filters with Single Bit Coefficients," ICASSP-84, San Diego, Calif., pp. 11.11.1-11.11.3., March 19-21, 1984.

[6] A. Fam, "Space-Time Duality in Digital Filter Structures," IEEE Trans. on Acoustics, Speech and Signal Processing, Vol. ASSP-31, No. 3, pp. 550-556, June 1983.

[7] J. Ullman, Computational Aspects of VLSI, Computer Science Press, Inc. 1984.

Figure Captions

Fig. 1. The minimal number of operations C as a function of N. Here r is treated as an integer parameter in the generation of this piecewise linear curve. The optimal values of r are indicated in the corresponding ranges of N. The behavior of the input partitioning algorithm is obtained via duality by just replacing N by M above. In this architecture, the computation of all the outputs is completed after a time corresponding to r-1 additions, but some of the outputs are computed sooner.

Fig. 2. A structured adder tree architecture for a pipelined implementation of the output partitioning algorithm. Several memory-adders are served with one such tree. The outputs from the memory-adders are multiplexed to the tree via a bank of serial in/ parallel out shift registers. The depicted case is for r = 4. The data at the rightmost position has an address tag corresponding to 1, while the leftmost one corresponds to 15. In such an architecture, all the outputs are available symultaneously after a time corresponding to r-1 additions.

Fig. 3. A pipelined architecture for computing all the partial sums of a set of numbers. The nested arrangement shown illustrates how this architecture is based on the recursion of (3.3). The depicted case corresponds to r = 3, where the three partial sums of the first two inputs are added to the third input to result in all the required seven partial sums.

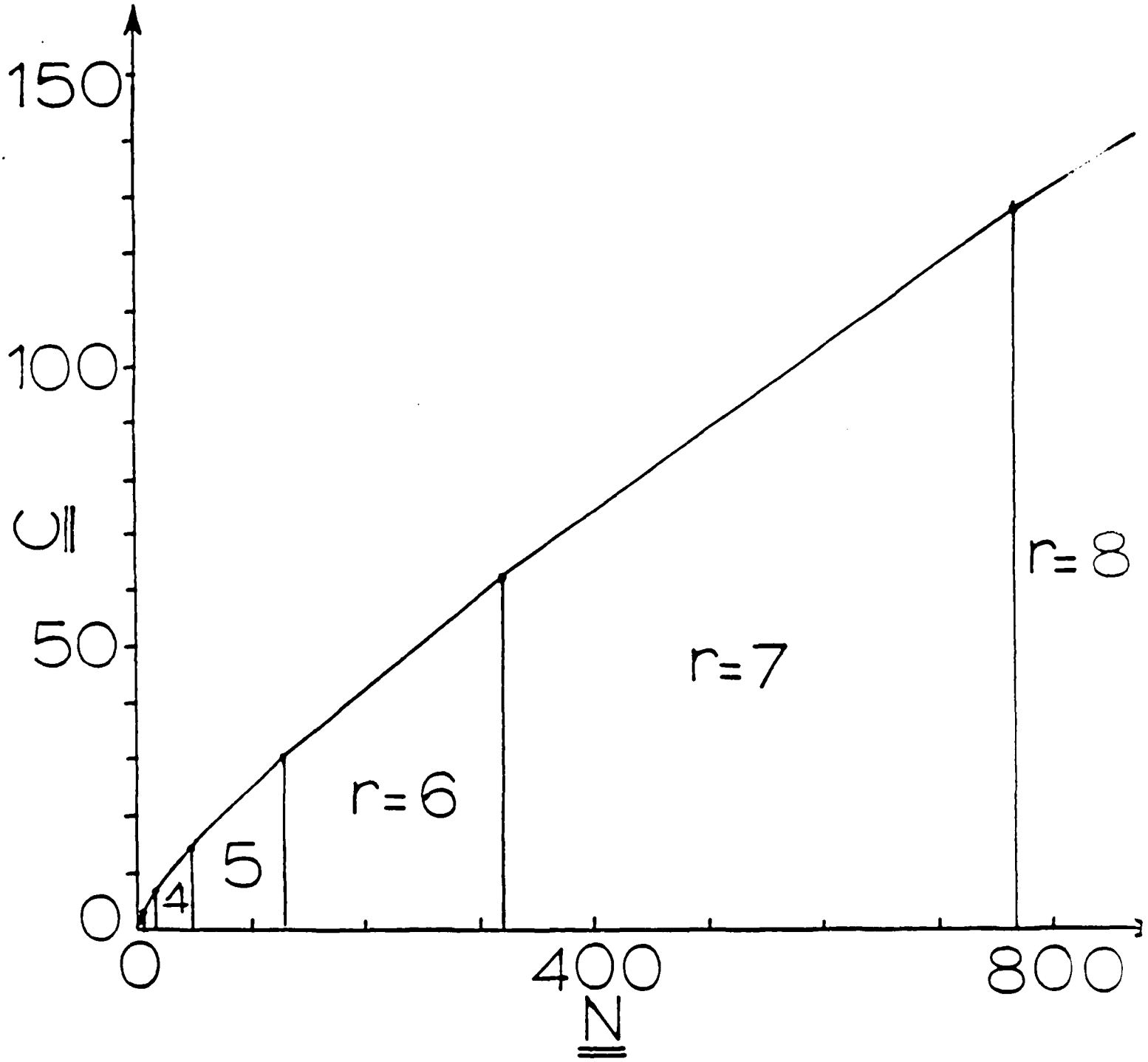


Fig. 1.

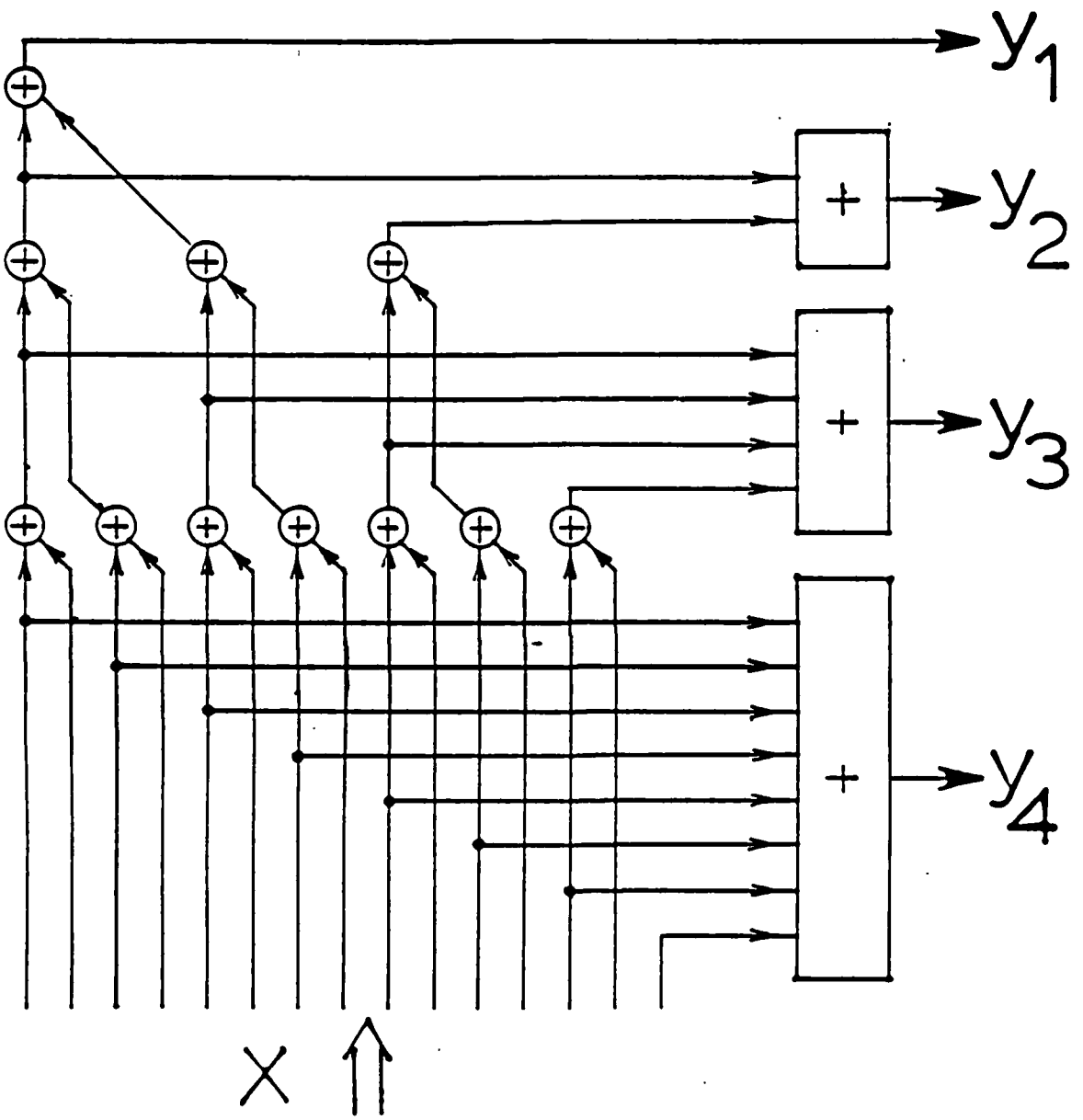


Fig. 2.

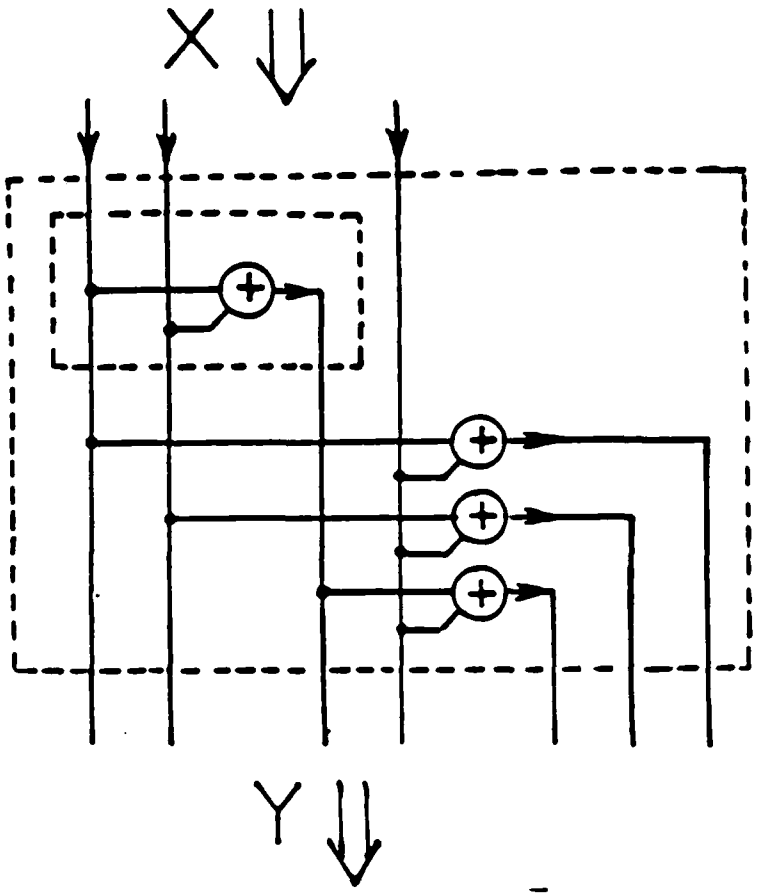


Fig. 3.

MISSION
of
Rome Air Development Center

RADC plans and executes research, development, test and selected acquisition programs in support of Command, Control, Communications and Intelligence (C^3I) activities. Technical and engineering support within areas of competence is provided to ESD Program Offices (POs) and other ESD elements to perform effective acquisition of C^3I systems. The areas of technical competence include communications, command and control, battle management, information processing, surveillance sensors, intelligence data collection and handling, solid state sciences, electromagnetics, and propagation, and electronic, maintainability, and compatibility.

END

5-87

DTIC